本节内容

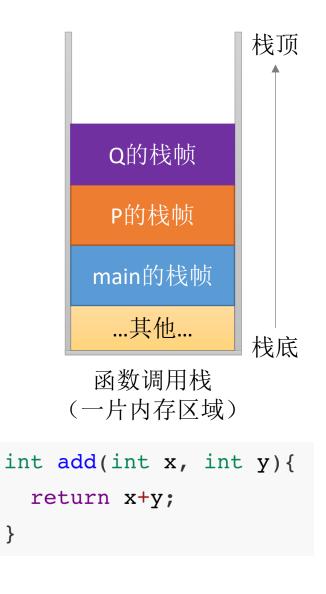
函数调用

机器级表示

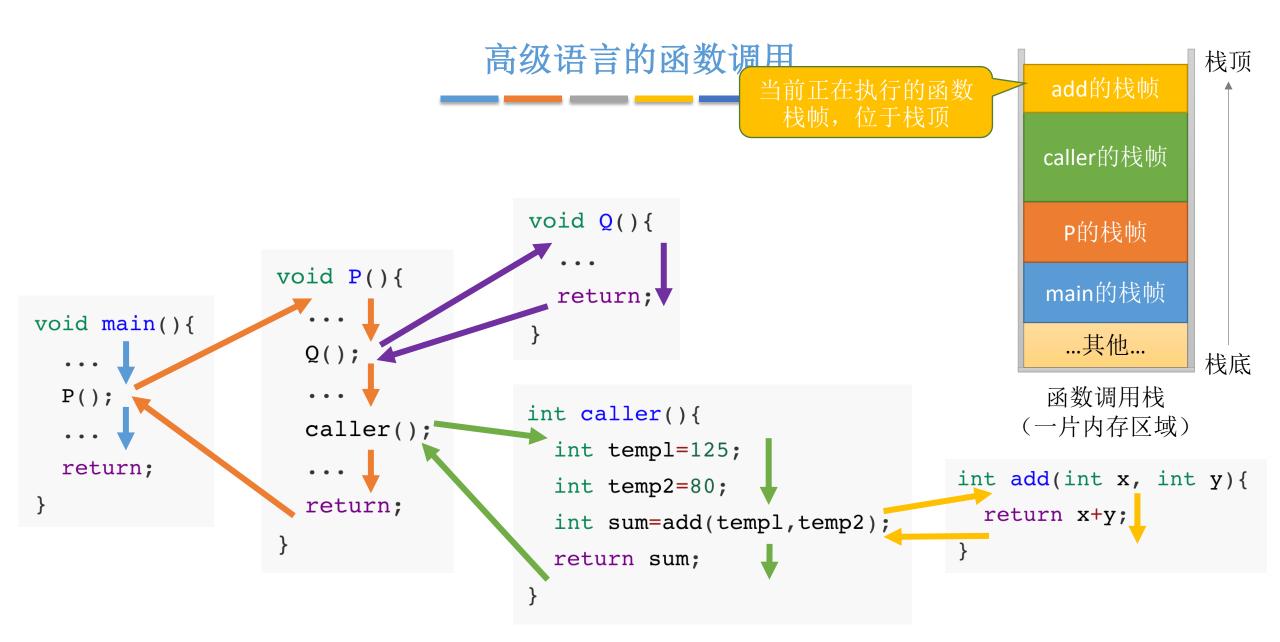
-- call、ret 指令

高级语言的函数调用

```
void Q(){
                   void P(){
                                         return;
void main(){
                     Q();
  P();
                                      int caller(){
                     caller();
  . . .
                                         int templ=125;
  return;
                                         int temp2=80;
                     return;
                                         int sum=add(templ,temp2);
                                         return sum;
```



函数的栈帧(Stack Frame):保存函数大括号内定义的局部变量、保存函数调用相关的信息



函数的栈帧(Stack Frame):保存函数大括号内定义的局部变量、保存函数调用相关的信息

函数调用指令: call <函数名>

函数返回指令: ret

x86汇编语言的函数调用

```
caller:
push ebp
mov ebp, esp
sub esp,24
mov [ebp-12],125
mov [ebp-8],80
mov eax, [ebp-8]
mov [esp+4],eax
mov eax, [ebp-12]
mov [esp],eax
call add
mov [ebp-4],eax
mov eax, [ebp-4]
leave
ret
```

```
add:
push ebp

mov ebp,esp

mov eax,[ebp+12]

mov edx,[ebp+8]

add eax,edx

leave

ret
```

```
int caller(){
  int templ=125;
  int temp2=80;
  int sum=add(templ,temp2);
  return sum;
}
```

```
dd的栈帧

caller的栈帧

P的栈帧

main的栈帧

...其他...

栈底
```

```
int add(int x, int y){
  return x+y;
}
```

call、ret指令

注: x86处理器中程序计数器 PC (Program Counter)通常被称为 IP (Instruction Pointer)

```
caller:
       push ebp
       mov ebp, esp
       sub esp,24
       mov [ebp-12],125
       mov [ebp-8],80
       mov eax, [ebp-8]
       mov [esp+4],eax
       mov eax, [ebp-12]
       mov [esp],eax
       call add
IP旧值→ mov [ebp-4],eax
       mov eax, [ebp-4]
       leave
```

ret

add:

push ebp

mov ebp,esp

mov eax,[ebp+12]

mov edx,[ebp+8]

add eax,edx

leave

ret

dd的栈帧
IP旧值(返回地址)
caller的栈帧
P的栈帧
main的栈帧
...其他...

栈底

call 指令的作用:

函数返回指令:

①将<mark>IP旧值</mark>压栈保存(保存在函数的栈帧顶部)

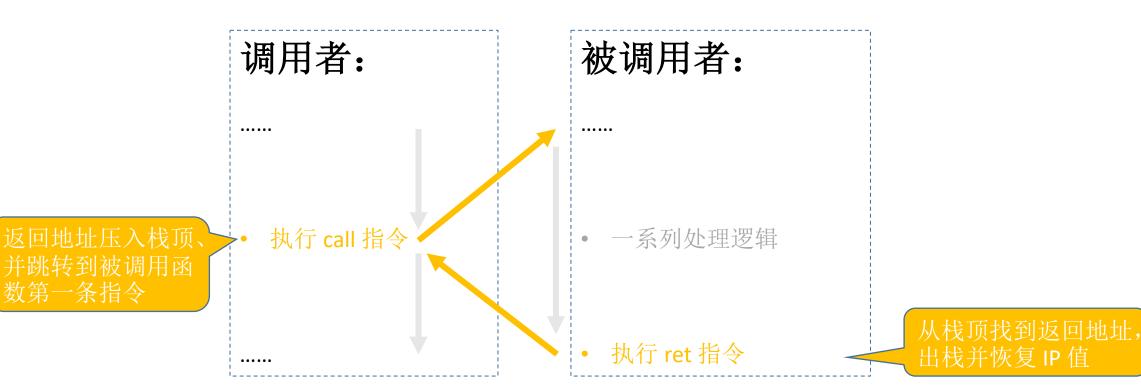
函数调用指令: call <函数名>

②设置<mark>IP新值</mark>,无条件转移至被调用函数的第一条指令

ret 指令的作用:

从函数的栈帧顶部找到IP旧值,将其出栈并恢复IP寄存器

总结: 函数调用的机器级表示





call 指令的作用:

- ①将IP旧值压栈保存(保存在函数的栈帧顶部)
- ②设置IP新值,无条件转移至被调用函数的第一条指令

ret 指令的作用:

从函数的栈帧顶部找到IP旧值,将其出栈并恢复IP寄存器

小朋友, 你是否有很多问号?

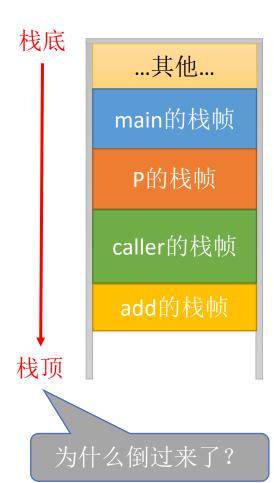
```
int caller(){
  int temp1=125;
  int temp2=80;
  int sum=add(temp1, temp2);
  return sum;
}
```



如何传递调用参数、返回值?

如何访问栈帧里的数据?

栈帧内可能包含哪些内容?





本节内容

函数调用

机器级表示

--如何访问栈帧?

小朋友, 你是否有很多问号?

```
int caller(){
  int templ=125;
  int temp2=80;
  int sum=add(templ,temp2);
  return sum;
}
```



如何传递调用参数、返回值?

如何访问栈帧里的数据?

栈帧内可能包含哪些内容?



栈为什么倒过来了?

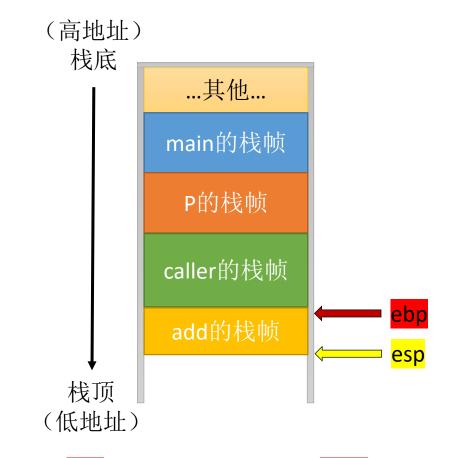


函数调用栈在内存中的位置 32位系统,进程虚拟地址空间为4GB OxFFFF FFFF (高地址) 内核代码、内核数据 栈底 操作系统内核区 1GB 结构等 ...其他... 栈底(高地址) 0xC000 0000 main的栈帧 用户栈 (Stack) 栈顶(低地址) P的栈帧 caller的栈帧 共享库的存储映射 printf() 函数 X add的栈帧 0x4000 0000 用尸区 **3GB-**栈顶 malloc 分配的区域 堆 (heap) (低地址) 读/写数据 全局变量、 静态变量 只读代码/数据 注: 大多数教材的图示,通常栈底在上,栈顶在下 程序指令、只读数据 0x0804 8000 未使用区

0x0000 0000

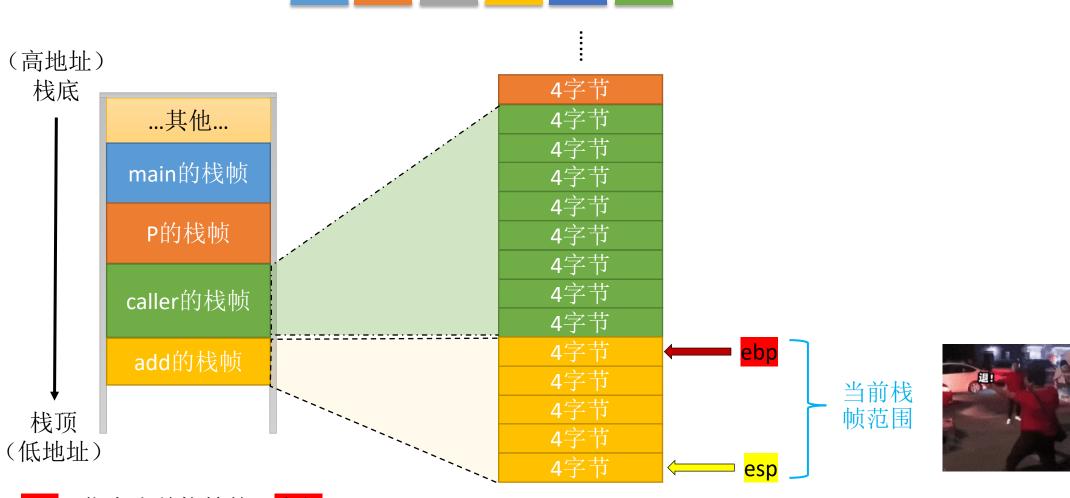






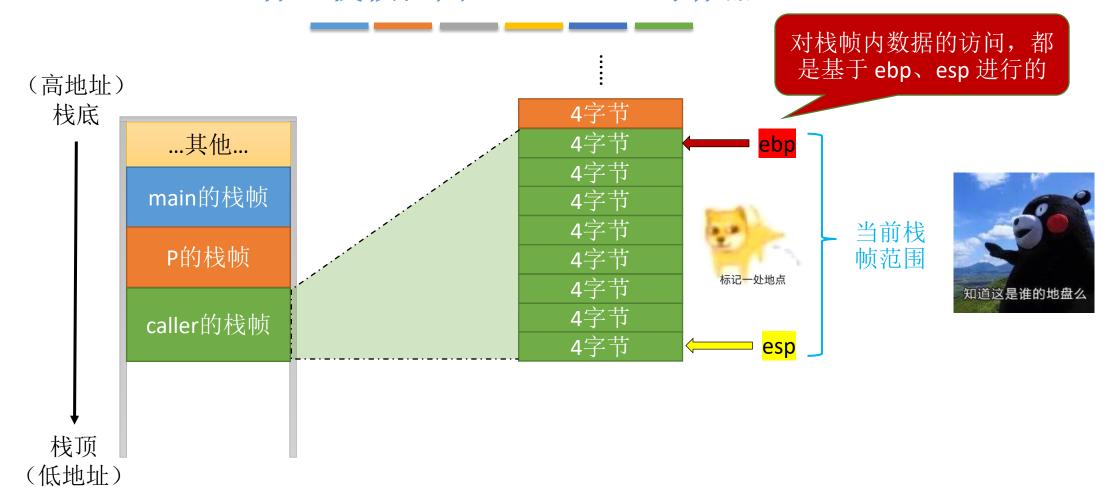
ebp: 指向当前栈帧的"底部"

esp: 指向当前栈帧的"<mark>顶部</mark>"



ebp: 指向当前栈帧的"<mark>底部</mark>"

esp: 指向当前栈帧的"<mark>顶部</mark>"



ebp: 指向当前栈帧的"<mark>底部</mark>"

esp: 指向当前栈帧的"<mark>顶部</mark>"

eax寄存器: 211

push、pop 指令实现入栈、出栈操作,x86 默认 以4字节为单位。指令格式如下:

Push 🐠 Pop 🦓

// 先让esp减4,再将 🐶 压入 //栈顶元素出栈写入 為 , 再让 esp加4

注1: 👽 可以是立即数、寄存器、主存地址

注2: அ可以是寄存器、主存地址

例:

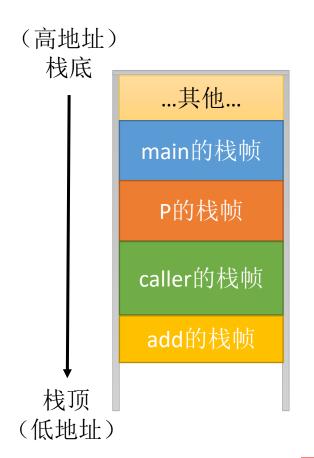
#将寄存器eax的值压栈 push eax

#将立即数985压栈 push 985

#将主存地址[ebp+8]里的数据压栈 push [ebp+8]

#栈顶元素出栈,写入寄存器eax pop eax

#栈顶元素出栈,写入主存地址[ebp+8] pop [ebp+8]



4字节 4字节 4字节 4字节 4字节 4字节 4字节 666 esp 4字节 4字节 4字节

指向当前栈帧的"底部"

指向当前栈帧的"<mark>顶部</mark>"

eax寄存器: 211

push、pop 指令实现入栈、出栈操作,x86 默认 以4字节为单位。指令格式如下:

Push 🐠 // 先让**esp减4**,再将 🐼 压入 Pop 🦓 //栈顶元素出栈写入 為 , 再让 esp加4

注1: 👽 可以是立即数、寄存器、主存地址

注2: அ可以是寄存器、主存地址

例:

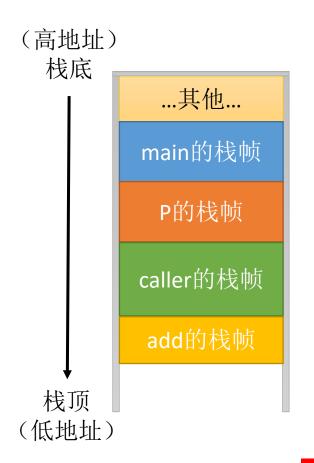
#将寄存器eax的值压栈 push eax

#将立即数985压栈 push 985

#将主存地址[ebp+8]里的数据压栈 push [ebp+8]

#栈顶元素出栈,写入寄存器eax pop eax

#栈顶元素出栈,写入主存地址[ebp+8] pop [ebp+8]



4字节 4字节 4字节 4字节 4字节 4字节 4字节 666 < 211 \cdot 4字节 4字节

指向当前栈帧的"底部"

指向当前栈帧的"<mark>顶部</mark>"

eax寄存器: 211

push、pop 指令实现入栈、出栈操作, x86 默认 以4字节为单位。指令格式如下:

Push 🧼 // 先让**esp减4**,再将 🐼 压入 Pop 🧆 //栈顶元素出栈写入 為 , 再让 esp加4

注1: 👽 可以是立即数、寄存器、主存地址

注2: அ可以是寄存器、主存地址

例:

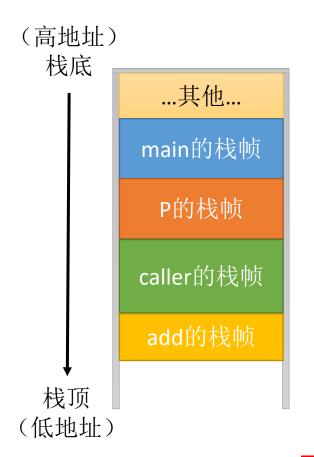
#将寄存器eax的值压栈 push eax

#将立即数985压栈 **push** 985

#将主存地址[ebp+8]里的数据压栈 push [ebp+8]

#栈顶元素出栈,写入寄存器eax pop eax

#栈顶元素出栈,写入主存地址[ebp+8] pop [ebp+8]



4字节 4字节 4字节 4字节 4字节 4字节 4字节 666 ື[ebp+8] [ebp+4] < 211 \cdot 985 esp 4字节

指向当前栈帧的"底部"

指向当前栈帧的"顶部"

push、pop 指令实现入栈、出栈操作,x86 默认 以4字节为单位。指令格式如下:

Push 🐠 // 先让**esp减4**,再将 🐼 压入 Pop 🧆 //栈顶元素出栈写入 🔷 , 再让 esp加4

注1: 👽 可以是立即数、寄存器、主存地址

注2: அ可以是寄存器、主存地址

例:

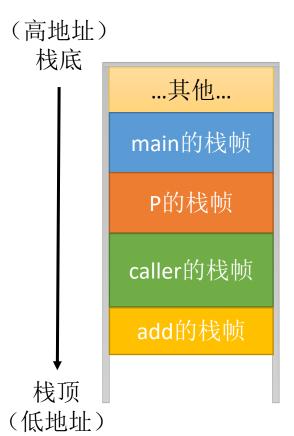
#将寄存器eax的值压栈 push eax

#将立即数985压栈 push 985

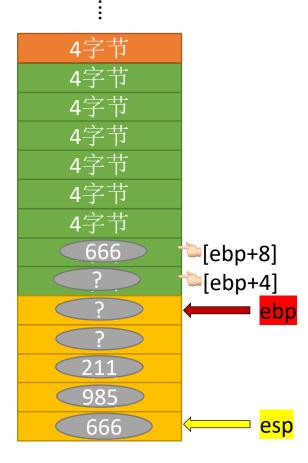
#将主存地址[ebp+8]里的数据压栈 push [ebp+8]

#栈顶元素出栈,写入寄存器eax pop eax

#栈顶元素出栈,写入主存地址[ebp+8] pop [ebp+8]



eax寄存器: 211



指向当前栈帧的"底部"

指向当前栈帧的"顶部"

eax寄存器: < 666

push、pop 指令实现入栈、出栈操作, x86 默认 以4字节为单位。指令格式如下:

Push 🧼 // 先让esp减4,再将 🐶 压入 Pop 🧆 //栈顶元素出栈写入 🖨 , 再让 esp加4

注1: 👽 可以是立即数、寄存器、主存地址

注2: அ可以是寄存器、主存地址

例:

#将寄存器eax的值压栈 push eax

push 985

push [ebp+8]

#将立即数985压栈

#将主存地址[ebp+8]里的数据压栈

(高地址) 栈底 ...其他... main的栈帧 P的栈帧 caller的栈帧 add的栈帧 栈顶 (低地址)

4字节 4字节 4字节 4字节 4字节 4字节 4字节 666 ື[ebp+8] [ebp+4] < 211 \cdot 985 esp 4字节

指向当前栈帧的"底部"

指向当前栈帧的"顶部"

王道考研/CSKAOYAN.COM

pop eax

#栈顶元素出栈,写入寄存器eax

#栈顶元素出栈,写入主存地址[ebp+8] pop [ebp+8]

eax寄存器: < 666

push、pop 指令实现入栈、出栈操作, x86 默认 以4字节为单位。指令格式如下:

Push 🧼 // 先让esp减4,再将 🐶 压入 //栈顶元素出栈写入 為 , 再让 esp加4 Pop 🗳

注1: 👽 可以是立即数、寄存器、主存地址

注2: அ可以是寄存器、主存地址

例:

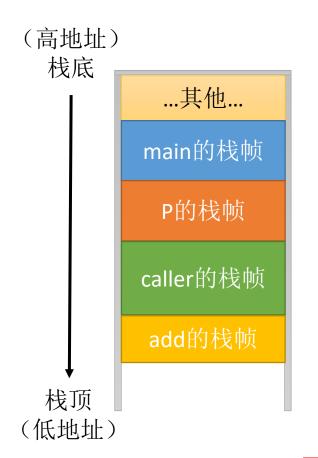
#将寄存器eax的值压栈 push eax

#将立即数985压栈 push 985

#将主存地址[ebp+8]里的数据压栈 push [ebp+8]

#栈顶元素出栈,写入寄存器eax pop eax

#栈顶元素出栈,写入主存地址[ebp+8] pop [ebp+8]



4字节 4字节 4字节 4字节 4字节 4字节 4字节 985 ື[ebp+8] [ebp+4] < 211 \cdot esp 4字节 4字节

指向当前栈帧的"底部"

指向当前栈帧的"顶部"

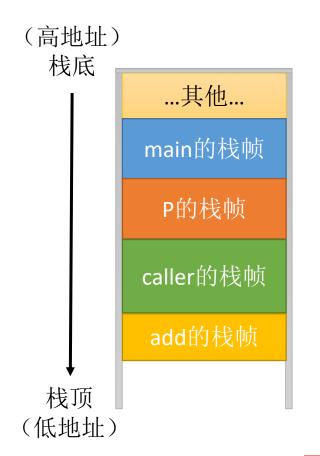
访问栈帧数据: mov 指令

eax寄存器: 211

例:

sub esp, 12 mov [esp+8], eax mov [esp+4], 958 mov eax, [ebp+8] mov [esp],eax add esp, 8

#栈顶指针-12 #将eax的值复制到主存[esp+8] #将985复制到主存[esp+4] #将主存[ebp+8]的值复制到eax #将eax的值复制到主存[esp] #栈顶指针+8



4字节 4字节 4字节 4字节 4字节 4字节 4字节 666 ebp esp 4字节 4字节 4字节

指向当前栈帧的"底部"

指向当前栈帧的"顶部"

例:

sub esp, 12 mov [esp+8], eax

mov [esp+4], 958

mov eax, [ebp+8]

mov [esp],eax add esp, 8

#栈顶指针-12

#将eax的值复制到主存[esp+8]

#将985复制到主存[esp+4]

#将主存[ebp+8]的值复制到eax

#将eax的值复制到主存[esp]

#栈顶指针+8



4字节 4字节 4字节 4字节 4字节 4字节 4字节 666 **→**[ebp+8] ebp 211 **→**[esp+8] 985 **□**[esp+4] 666

指向当前栈帧的"底部"

指向当前栈帧的"顶部"

例:

sub esp, 12 mov [esp+8], eax mov [esp+4], 958 mov eax, [ebp+8] mov [esp],eax

add esp, 8

#栈顶指针-12

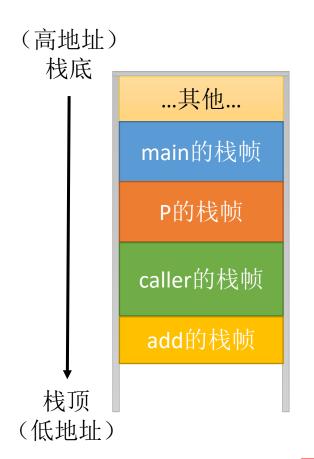
#将eax的值复制到主存[esp+8]

#将985复制到主存[esp+4]

#将主存[ebp+8]的值复制到eax

#将eax的值复制到主存[esp]

#栈顶指针+8



4字节 4字节 4字节 4字节 4字节 4字节 4字节 666 211 esp 4字节 +8 4字节 esp

可以用 mov 指令,结合 esp、ebp 指针访问栈帧数据

可以用减法/加法指令,即 sub/add 修改栈顶指针 esp 的值

指向当前栈帧的"底部"

指向当前栈帧的"顶部"

总结:如何访问栈帧?

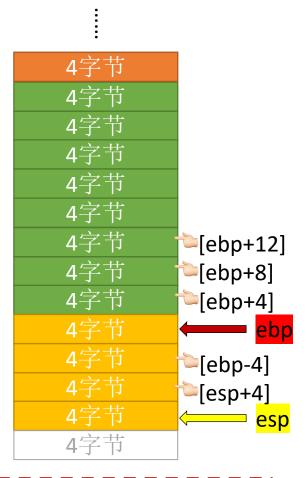
方法一:

```
Push → // 先让esp减4,再将 → 压入 // 栈顶元素出栈写入 → ,再让 esp加4 注1: → 可以是立即数、寄存器、主存地址注2: → 可以是寄存器、主存地址
```

方法二:

mov 指令,结合 esp、ebp 指针访问栈帧数据

注:可以用减法/加法指令,即 sub/add 修改栈顶指针 esp 的值



ebp:指向当前栈帧的"底部"

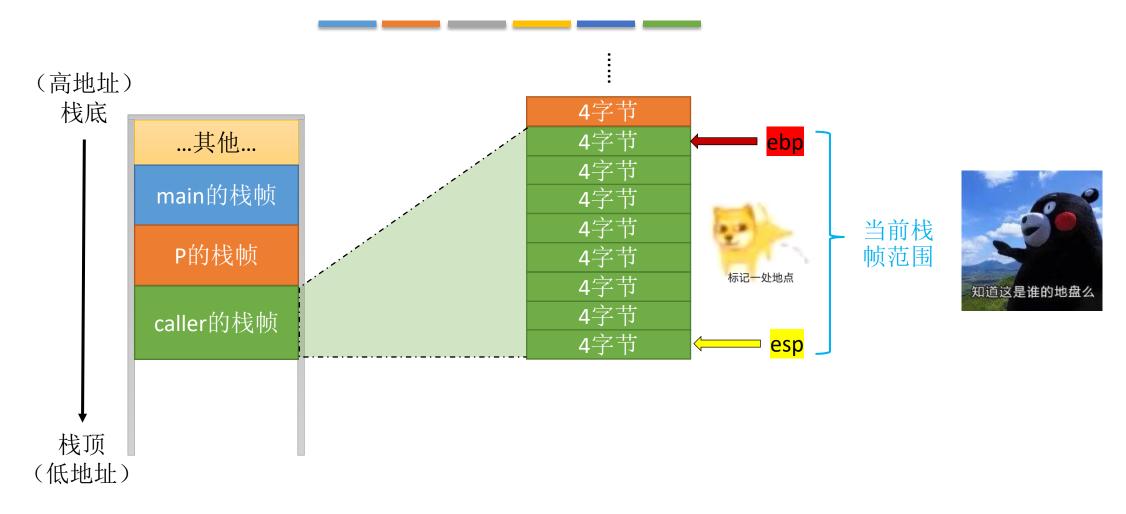
esp: 指向当前栈帧的"顶部"

本节内容

函数调用

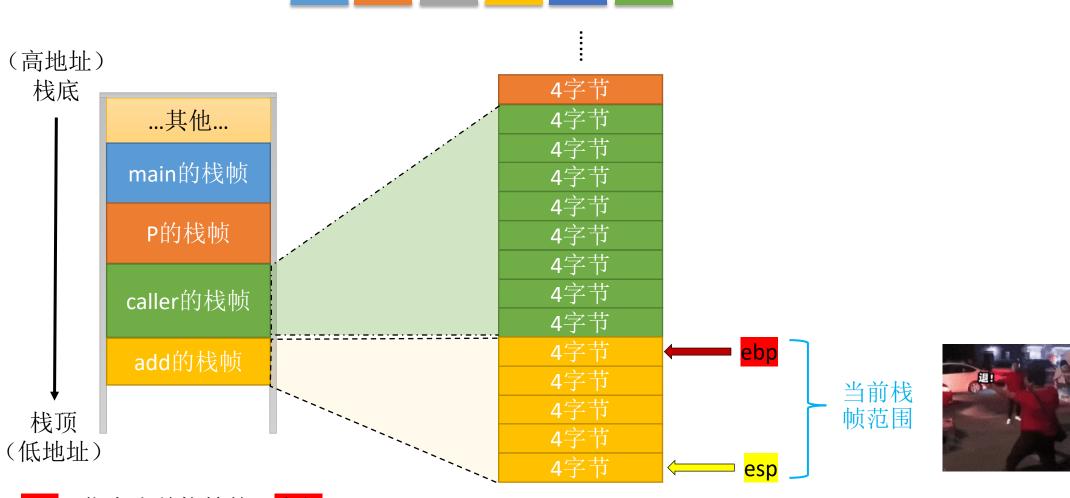
机器级表示

--如何切换栈帧?



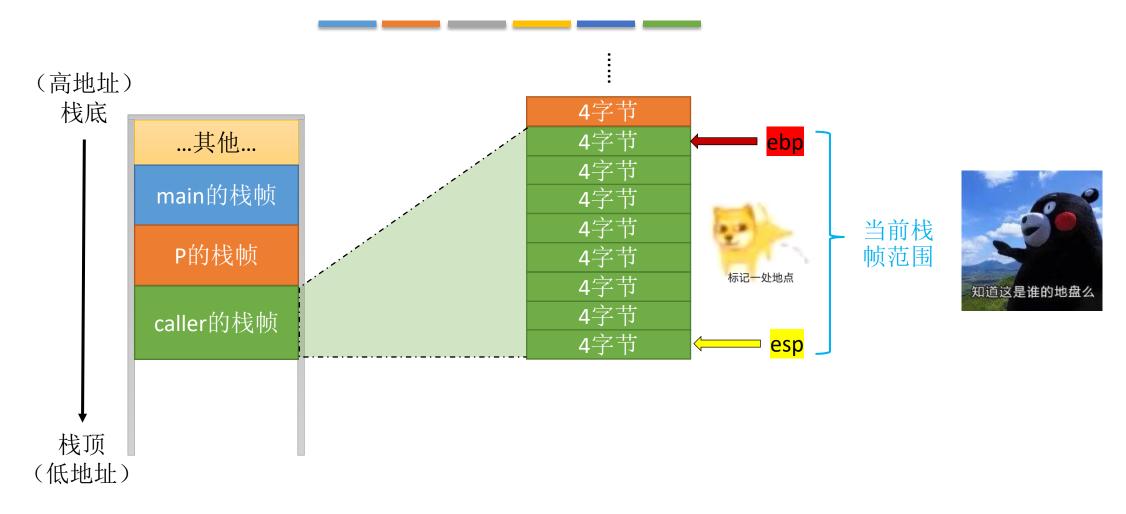
ebp: 指向当前栈帧的"底部"

esp: 指向当前栈帧的"<mark>顶部</mark>"



ebp: 指向当前栈帧的"<mark>底部</mark>"

esp: 指向当前栈帧的"<mark>顶部</mark>"



ebp: 指向当前栈帧的"底部"

esp: 指向当前栈帧的"<mark>顶部</mark>"

```
caller:
push ebp
mov ebp, esp
sub esp,24
mov [ebp-12],125
mov [ebp-8],80
mov eax, [ebp-8]
mov [esp+4],eax
mov eax, [ebp-12]
mov [esp],eax
call add
mov [ebp-4],eax
mov eax, [ebp-4]
leave
ret
```

add:

push ebp

mov ebp,esp

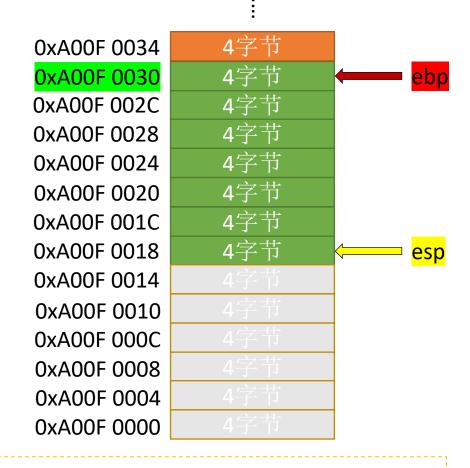
mov eax,[ebp+12]

mov edx,[ebp+8]

add eax,edx

leave

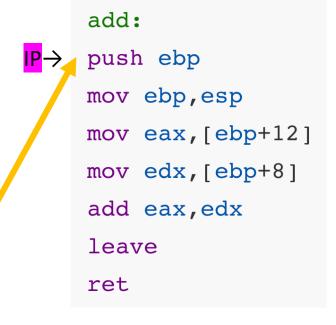
ret

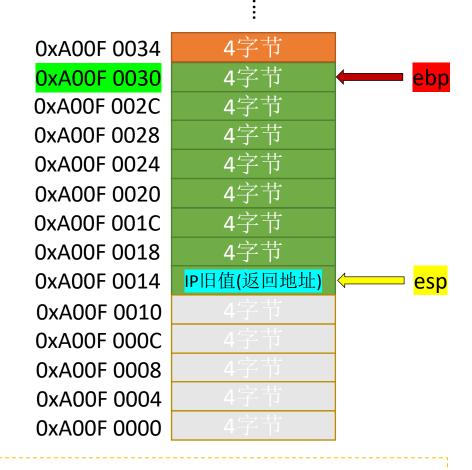


call 指令的作用:

- ①将IP旧值压栈保存(<mark>效果相当于 push IP</mark>)
- ②设置IP新值,无条件转移至被调用函数的第一条指令(<mark>效果相当于 jmp add</mark>)

```
caller:
push ebp
mov ebp, esp
sub esp,24
mov [ebp-12],125
mov [ebp-8],80
mov eax, [ebp-8]
mov [esp+4],eax
mov eax, [ebp-12]
mov [esp],eax
call add
mov [ebp-4],eax
mov eax, [ebp-4]
leave
ret
```

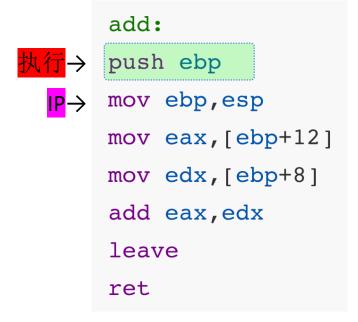


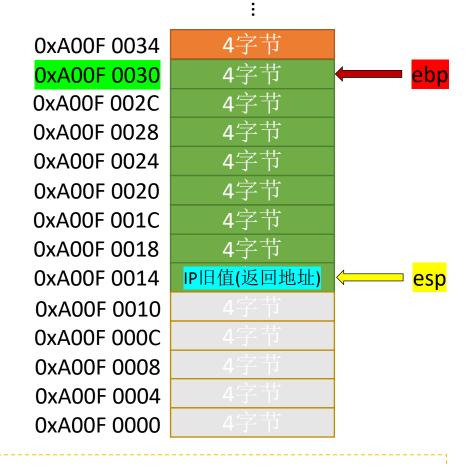


call 指令的作用:

- ①将IP旧值压栈保存(<mark>效果相当于 push IP</mark>)
- ②设置IP新值,无条件转移至被调用函数的第一条指令(<mark>效果相当于 jmp add</mark>)

```
caller:
push ebp
mov ebp, esp
sub esp,24
mov [ebp-12],125
mov [ebp-8],80
mov eax, [ebp-8]
mov [esp+4],eax
mov eax, [ebp-12]
mov [esp],eax
call add
mov [ebp-4],eax
mov eax, [ebp-4]
leave
ret
```



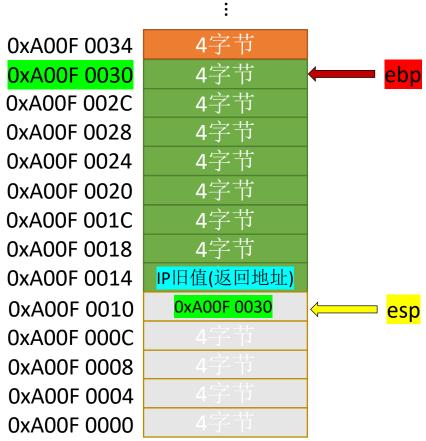


call 指令的作用:

- ①将IP旧值压栈保存(<mark>效果相当于 push IP</mark>)
- ②设置IP新值,无条件转移至被调用函数的第一条指令(<mark>效果相当于 jmp add</mark>)

```
caller:
push ebp
mov ebp, esp
sub esp,24
mov [ebp-12],125
mov [ebp-8],80
mov eax, [ebp-8]
mov [esp+4],eax
mov eax, [ebp-12]
mov [esp],eax
call add
mov [ebp-4],eax
mov eax, [ebp-4]
leave
ret
```

```
add:
push ebp
mov ebp, esp
mov eax,[ebp+12]
mov edx,[ebp+8]
add eax, edx
leave
ret
```



```
caller:
push ebp
mov ebp, esp
sub esp,24
mov [ebp-12],125
mov [ebp-8],80
mov eax, [ebp-8]
mov [esp+4],eax
mov eax, [ebp-12]
mov [esp],eax
call add
mov [ebp-4],eax
mov eax, [ebp-4]
leave
ret
```

```
add:

push ebp

mov ebp,esp

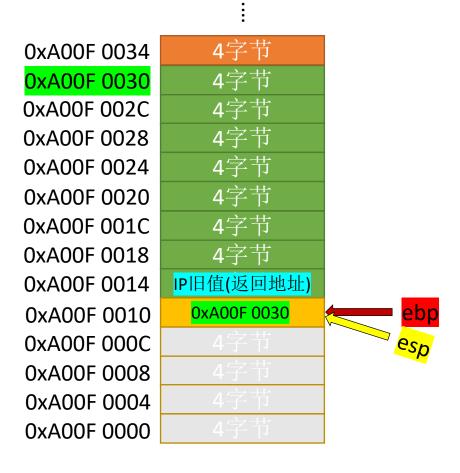
mov eax,[ebp+12]

mov edx,[ebp+8]

add eax,edx

leave

ret
```

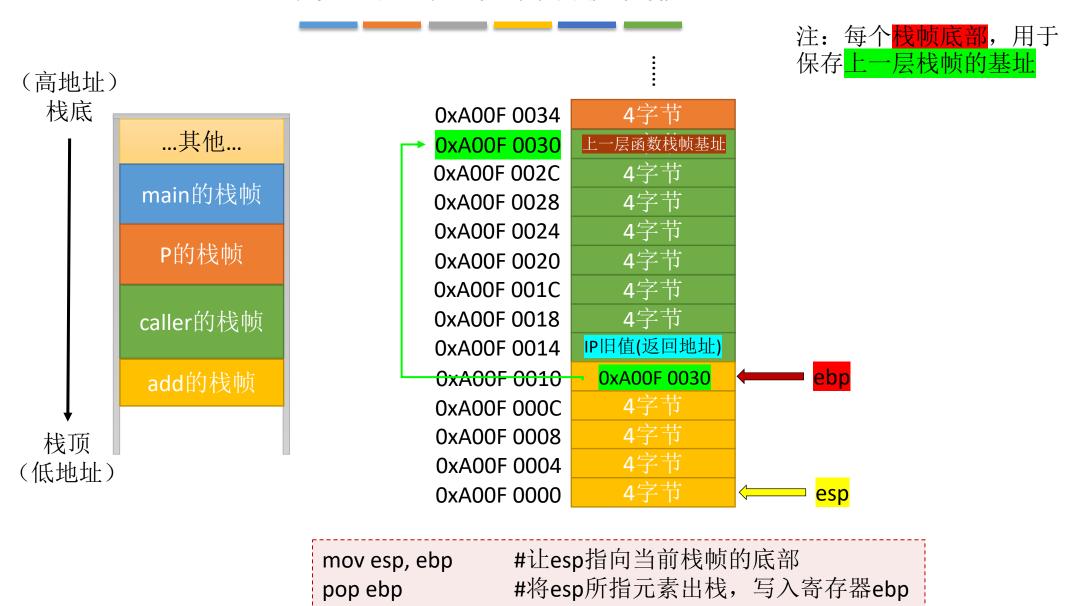


在每个函数开头

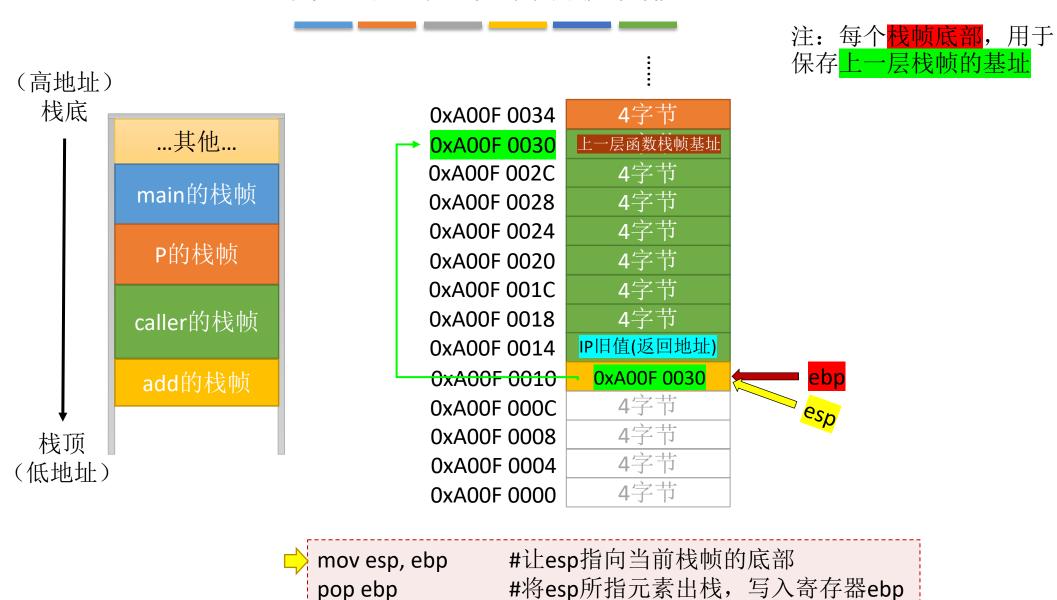
函数调用时,如何切换栈帧?

```
caller:
                                      add:
                                                               0xA00F 0034
       push ebp
                                      push ebp
                                                               0xA00F 0030
       mov ebp, esp
                                      mov ebp, esp
                                                               0xA00F 002C
                                                               0xA00F 0028
       sub esp,24
                                      mov eax,[ebp+12]
                                                               0xA00F 0024
       mov [ebp-12],125
                                      mov edx,[ebp+8]
                                                               0xA00F 0020
       mov [ebp-8],80
                                      add eax, edx
                                                               0xA00F 001C
       mov eax, [ebp-8]
                                                               0xA00F 0018
                                      leave
                                                               0xA00F 0014
       mov [esp+4],eax
                                      ret
                                                               0xA00F 0010
       mov eax, [ebp-12]
                                                               0xA00F 000C
       mov [esp],eax
                                                               0xA00F 0008
                                  指令: enter
       call add
                                                               0xA00F 0004
                                                               0xA00F 0000
IP旧值→ mov [ebp-4],eax
       mov eax, [ebp-4]
       leave
                                                #保存上一层函数的栈帧基址(ebp旧值)
                                  push ebp
                                                #设置当前函数的栈帧基址(ebp新值)
                                  mov ebp,esp
       ret
```

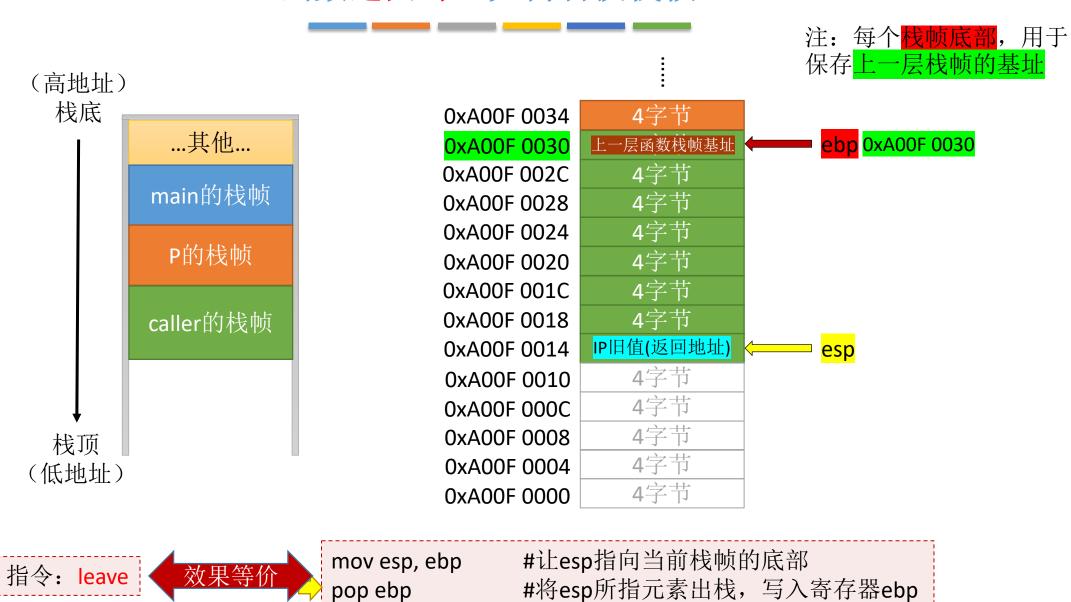
函数返回时,如何切换栈帧?



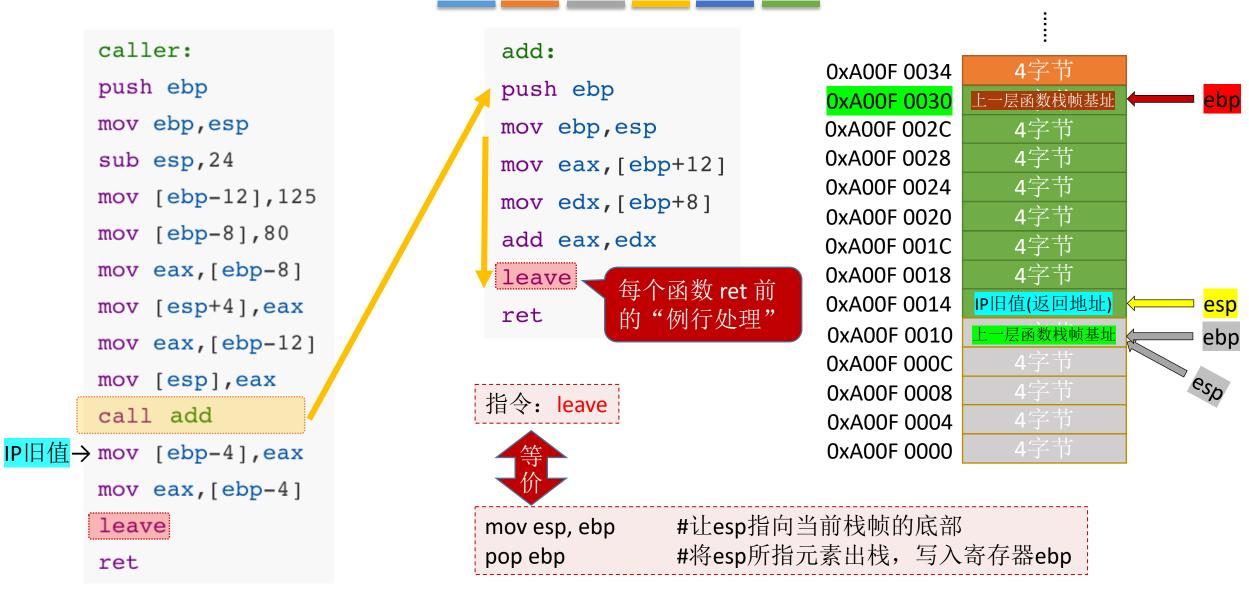
函数返回时,如何切换栈帧?



函数返回时,如何切换栈帧?



函数返回时,如何切换栈帧?



函数返回时,如何切换栈帧?



总结: 函数调用的机器级表示

除了main函数,其他所有函数的汇编代码结构都一样!



调用者:

.....

返回地址压入栈顶、 并跳转到被调用函 数第一条指令 执行 call 指令

被调用者:

保存上一层函数栈帧, 设置当前函数栈帧

一系列处理逻辑

• 恢复上一层函数的栈帧

执行 ret 指令

push ebp mov ebp,esp

或: enter指令

mov esp, ebp pop ebp

或: leave指令

从栈顶找到返回地址 出栈并恢复 IP 值

一个函数的汇编代码框架:

被调用者:

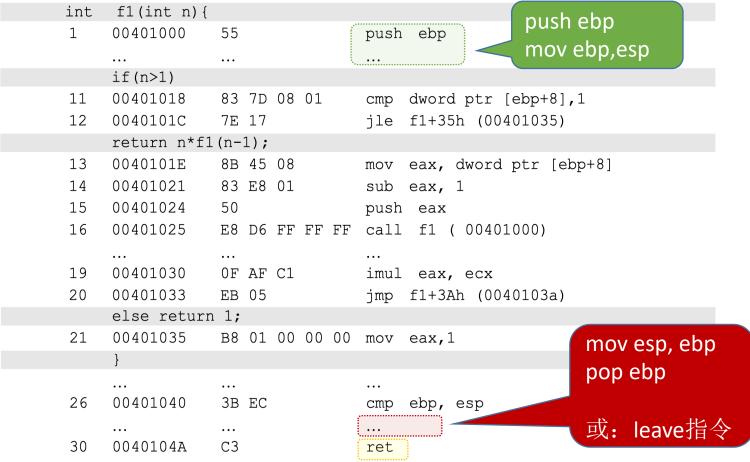
保存上一层函数栈帧, 设置当前函数栈帧

• 一系列处理逻辑

- 恢复上一层函数的栈帧
- 执行 ret 指令

例: 2019年真题

45. (16 分) 已知 $f(n) = n! = n \times (n-1) \times (n-2) \times \cdots \times 2 \times 1$,计算 f(n)的 C 语言函数 f1 的源程序(阴影部分)及其在 32 位计算机 M 上的部分机器级代码如下:



其中,机器级代码行包括行号、虚拟地址、机器指令和汇编指令,计算机 M 按字节编址,int型数据占32位。请回答下列问题:

一个函数的汇编代码框架:

被调用者:

保存上一层函数栈帧, 设置当前函数栈帧

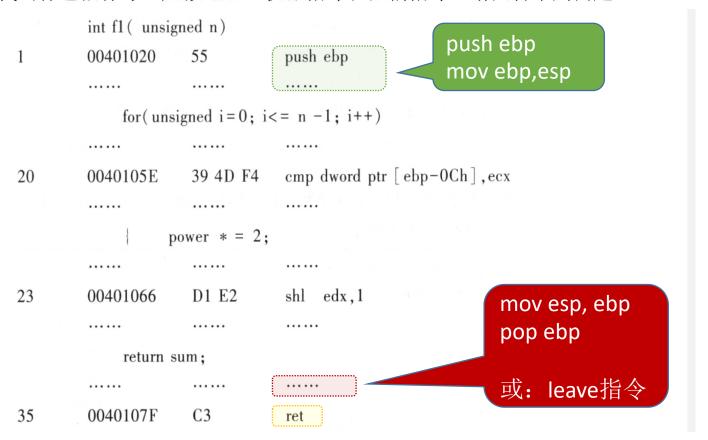
• 一系列处理逻辑

- 恢复上一层函数的栈帧
- 执行 ret 指令

例: 2017年真题

44. (10 分) 在按字节编址的计算机 M 上, 题 43 中 f1 的部分源程序(阴影部分) 与对应的机器级代码(包括指令的虚拟地址) 如下:

其中,机器级代码行包括行号、虚拟地址、机器指令和汇编指令。请回答下列问题。



本节内容

函数调用

机器级表示

- --栈帧内包含哪些内容?
- --参数、返回值传递

问题回顾

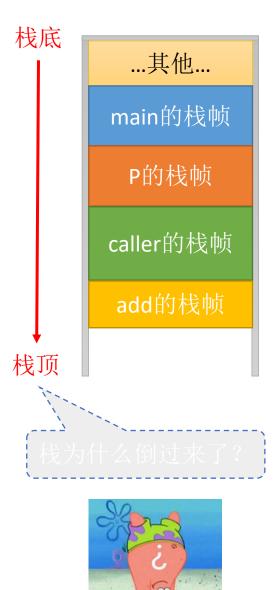
```
int caller(){
  int temp1=125;
  int temp2=80;
  int sum=add(temp1, temp2);
  return sum;
}
```



如何传递调用参数、返回值?

如何访问栈帧里的数据?

栈帧内可能包含哪些内容?



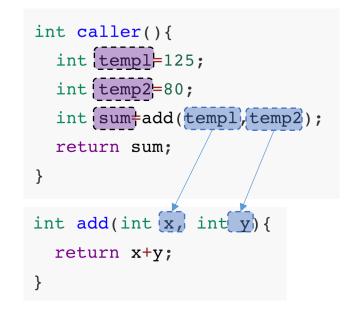


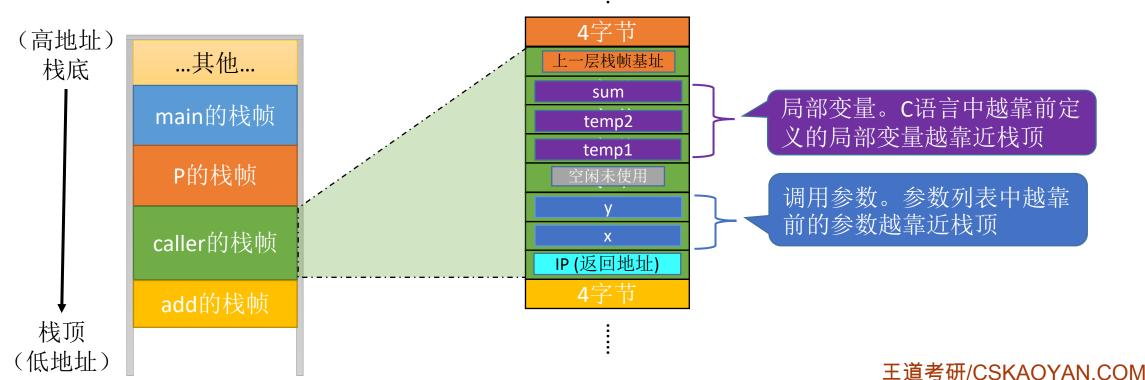


GNU/Linux家的好东西, 开源、自由、非常流行

一个栈帧内可能包含哪些内容?

- gcc 编译器将每个栈帧大小设置为 16B 的整数倍(当前函数的栈帧除外), 因此栈帧内可能出现空闲未使用的区域。
- 通常将局部变量集中存储在栈帧底部区域
- 通常将调用参数集中存储在栈帧顶部区域
- 栈帧最底部一定是上一层栈帧基址(ebp旧值)
- 栈帧最顶部一定是返回地址(当前函数的栈帧除外)





一个栈帧内可能包含哪些内容?

每个函数开头都要执行: IP(返回地址) push ebp 上一层栈帧基址 mov ebp,esp 局部变量。C语言中越靠前定 若干个局部变量 义的局部变量越靠近栈顶 栈帧 栈帧大小为16B整数倍,因此 结构 调用参数。参数列表中越靠前 若干个调用参数 的参数越靠近栈顶 IP(返回地址) 函数调用时, call 指令将IP寄 存器值(返回地址)压栈保存 上一层栈帧基址

一定存在。用于恢复上一层函数的 栈帧

不一定存在。有些函数可能不定义局部变量

不一定存在。如果其他部分刚好是 16B整数倍,则不会留下"零头"

不一定存在。有些函数调用不需要传参数

一定存在(发生调用时)。但凡调用其他函数,就必须记录返回地址

汇编代码实战

```
IP(返回地址)
    上一层栈帧基址
    若干个局部变量
栈
帧
结构
    未使用的区域
    若干个调用参数
     IP(返回地址)
```

```
caller:
push ebp
mov ebp, esp
sub esp,24
mov [ebp-12],125
mov [ebp-8],80
mov eax, [ebp-8]
mov [esp+4],eax
mov eax, [ebp-12]
mov [esp],eax
call add
mov [ebp-4],eax
mov eax, [ebp-4]
leave
ret
```

```
add:

push ebp

mov ebp,esp

mov eax,[ebp+12]

mov edx,[ebp+8]

add eax,edx

leave

ret
```

```
0xA00F 0058
                4字节
                             P函数
                             栈帧
0xA00F 0034
0xA00F 0030
              0xA00F 0058
0xA00F 002C
0xA00F 0028
0xA00F 0024
0xA00F 0020
0xA00F 001C
0xA00F 0018
0xA00F 0014
0xA00F 0010
0xA00F 000C
0xA00F 0008
```

```
int caller(){
  int templ=125;
  int temp2=80;
  int sum=add(templ,temp2);
  return sum;
}
```

```
eax: edx:
int add(int x, int y){
  return x+y;
}
```

访问当前函数的局部变量: [ebp-4]、[ebp-8]... 汇编代码实战:访问局部变量 4字节 0xA00F 0058 P函数 栈帧 caller: add: IP(返回地址) 0xA00F 0034 4字节 push ebp push ebp 上一层栈帧基址 0xA00F 0030 0xA00F 0058 — <mark>ebp</mark> mov ebp, esp mov ebp, esp 未初始化(sum) 0xA00F 002C [ebp-4] mov eax,[ebp+12] sub esp,24 0xA00F 0028 80 (temp2) [ebp-8] 若干个局部变量 0xA00F 0024 125 (temp1) mov edx, [ebp+8] mov [ebp-12],125 [ebp-12] 4字节 0xA00F 0020 栈 add eax, edx mov [ebp-8],80 4字节 0xA00F 001C 帧 leave mov eax, [ebp-8] 4字节 0xA00F 0018 esp 结 未使用的区域 ret mov [esp+4],eax 构 0xA00F 0014 mov eax, [ebp-12] 0xA00F 0010 0xA00F 000C mov [esp],eax 0xA00F 0008 若干个调用参数 call add mov [ebp-4],eax edx: eax: IP(返回地址) int caller(){ mov eax, [ebp-4] 上一层栈帧基址 int add(int x, int y){ int templ=125; leave int temp2=80; return x+y; ret int sum=add(templ,temp2); return sum;

汇编代码实战: 传递参数

```
IP(返回地址)
    上一层栈帧基址
    若干个局部变量
栈
帧
结
    若干个调用参数
     IP(返回地址)
    上一层栈帧基址
```

```
caller:
push ebp
mov ebp, esp
sub esp,24
mov [ebp-12],125
mov [ebp-8],80
mov eax, [ebp-8]
mov [esp+4],eax
mov eax, [ebp-12]
mov [esp],eax
call add
mov [ebp-4],eax
mov eax, [ebp-4]
leave
ret
```

```
add:

push ebp

mov ebp,esp

mov eax,[ebp+12]

mov edx,[ebp+8]

add eax,edx

leave

ret
```

```
4字节
0xA00F 0058
                              P函数
                              栈帧
0xA00F 0034
                 4字节
0xA00F 0030
             0xA00F 0058
                             — <mark>ebp</mark>
             未初始化(sum)
0xA00F 002C
                             ebp-4]
0xA00F 0028
               80 (temp2)
                             [ebp-8]
0xA00F 0024
              125 (temp1)
                             [ebp-12]
                 4字节
0xA00F 0020
0xA00F 001C
                 80 (y)
                             [esp+4]
0xA00F 0018
                125 (x)
                              esp
0xA00F 0014
0xA00F 0010
0xA00F 000C
0xA00F 0008
```

```
int caller(){
  int templ=125;
  int temp2=80;
  int sum=add(templ,temp2);
  return sum;
}
eax: 125 edx:

int add(int x, int y){
  return x+y;
  }

return sum;

}
```

IP(返回地址) 上一层栈帧基址 若干个局部变量 栈 帧 结 若干个调用参数 IP(返回地址) 上一层栈帧基址

汇编代码实战: 函数调用

```
caller:
push ebp
mov ebp, esp
sub esp,24
mov [ebp-12],125
mov [ebp-8],80
mov eax, [ebp-8]
mov [esp+4],eax
mov eax, [ebp-12]
mov [esp],eax
call add
mov [ebp-4],eax
mov eax, [ebp-4]
leave
ret
```

```
add:

push ebp

mov ebp,esp

mov eax,[ebp+12]

mov edx,[ebp+8]

add eax,edx

leave

ret
```

```
4字节
0xA00F 0058
                            P函数
                            栈帧
0xA00F 0034
                4字节
0xA00F 0030
            0xA00F 0058
                            ebp
            未初始化(sum)
0xA00F 002C
0xA00F 0028
              80 (temp2)
                            caller
0xA00F 0024
             125 (temp1)
                            栈帧
                4字节
0xA00F 0020
0xA00F 001C
               80 (y)
0xA00F 0018
               125 (x)
0xA00F 0014
              IP (返回地址)
                              esp
0xA00F 0010
0xA00F 000C
0xA00F 0008
```

```
int caller(){
  int templ=125;
  int temp2=80;
  int sum=add(templ,temp2);
  return sum;
}
eax: 125 edx:

int add(int x, int y){
  return x+y;
  }

return sum;

}
```

IP(返回地址) 上一层栈帧基址 若干个局部变量 栈 帧 结 若干个调用参数 IP(返回地址)

```
caller:
push ebp
mov ebp, esp
sub esp,24
mov [ebp-12],125
mov [ebp-8],80
mov eax, [ebp-8]
mov [esp+4],eax
mov eax, [ebp-12]
mov [esp],eax
call add
mov [ebp-4],eax
mov eax, [ebp-4]
leave
ret
```

```
汇编代码实战: 切换栈帧
                                                        4字节
                                         0xA00F 0058
                                                                    P函数
                                                                    栈帧
                   add:
                                         0xA00F 0034
                                                         4字节
                   push ebp
                                         0xA00F 0030
                                                     0xA00F 0058
                   mov ebp, esp
                                                     未初始化(sum)
                                         0xA00F 002C
                   mov eax,[ebp+12]
                                         0xA00F 0028
                                                       80 (temp2)
                                                                     caller
                                         0xA00F 0024
                                                      125 (temp1)
                   mov edx, [ebp+8]
                                                                     栈帧
                                                         4字节
                                         0xA00F 0020
                   add eax, edx
                                         0xA00F 001C
                                                        80 (y)
                   leave
                                         0xA00F 0018
                                                        125 (x)
                   ret
                                         0xA00F 0014
                                                       IP (返回地址)

■ ebp

                                         0xA00F 0010
                                                      0xA00F 0030
                                         0xA00F 000C
                                         0xA00F 0008
                                                            edx:
                                               eax:
                  int caller(){
                                               int add(int x, int y){
                    int templ=125;
                    int temp2=80;
                                                 return x+y;
                    int sum=add(templ,temp2);
                    return sum;
```

访问上一层函数传过来的参 [ebp+8] [ebp+12]...

IP(返回地址) 上一层栈帧基址 若干个局部变量 栈 帧 结 若干个调用参数 IP(返回地址)

汇编代码实战:访问参数

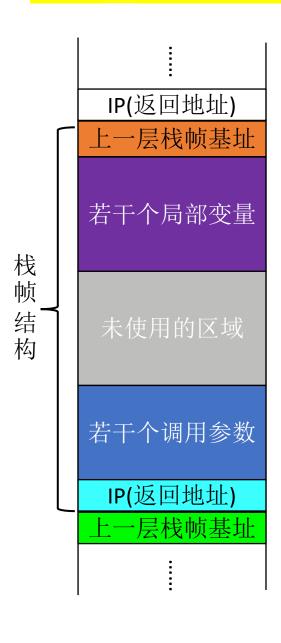
ret

```
caller:
push ebp
mov ebp, esp
sub esp,24
mov [ebp-12],125 ■
mov [ebp-8],80
mov eax, [ebp-8]
mov [esp+4],eax
mov eax, [ebp-12]
mov [esp],eax
call add
mov [ebp-4],eax
mov eax, [ebp-4]
leave
ret
```

```
0xA00F 0058
                                       4字节
                                                   P函数
                                                   栈帧
add:
                       0xA00F 0034
                                       4字节
push ebp
                      0xA00F 0030
                                   0xA00F 0058
mov ebp, esp
                                   未初始化(sum)
                      0xA00F 002C
mov eax,[ebp+12]
                      0xA00F 0028
                                     80 (temp2)
                                                   caller
                      0xA00F 0024
                                    125 (temp1)
mov edx, [ebp+8]
                                                   栈帧
                                       4字节
                      0xA00F 0020
add eax, edx
                      0xA00F 001C
                                      80 (y)
                                                   [ebp+12]
leave
                      0xA00F 0018
                                      125 (x)
                                                   [ebp+8]
                      0xA00F 0014
                                     IP (返回地址)
                                                   ebp
                      0xA00F 0010
                                    0xA00F 0030
                      0xA00F 000C
                      0xA00F 0008
```

```
edx: 125
                               eax:
int caller(){
                               int add(int x, int y){
  int templ=125;
  int temp2=80;
                                 return x+y;
  int sum=add(templ,temp2);
  return sum;
```

通常用eax寄存器返回结果



```
汇编代码实战: 传递返回值
                                                           4字节
                                            0xA00F 0058
                                                                      P函数
                                                                       栈帧
caller:
                       add:
                                            0xA00F 0034
                                                           4字节
push ebp
                       push ebp
                                            0xA00F 0030
                                                        0xA00F 0058
mov ebp, esp
                       mov ebp, esp
                                                        未初始化(sum)
                                            0xA00F 002C
                       mov eax,[ebp+12]
sub esp,24
                                            0xA00F 0028
                                                         80 (temp2)
                                                                       caller
                                            0xA00F 0024
                                                         125 (temp1)
                       mov edx, [ebp+8]
mov [ebp-12],125
                                                                       栈帧
                                                           4字节
                                            0xA00F 0020
                       add eax, edx
mov [ebp-8],80
                                            0xA00F 001C
                                                           80 (y)
                                                                      [ebp+12]
                       leave
mov eax, [ebp-8]
                                            0xA00F 0018
                                                          125 (x)
                                                                      [ebp+8]
                       ret
mov [esp+4],eax
                                            0xA00F 0014
                                                         IP (返回地址)
                                                                       ebp
mov eax, [ebp-12]
                                            0xA00F 0010
                                                         0xA00F 0030
                                            0xA00F 000C
mov [esp],eax
                                            0xA00F 0008
call add
mov [ebp-4],eax
                                                  eax:
                                                               edx: 125
                      int caller(){
mov eax, [ebp-4]
                                                  int add(int x, int y){
                        int templ=125;
leave
                        int temp2=80;
                                                    return x+y;
ret
                        int sum=add(templ,temp2);
                        return sum;
                                                  王道考研/CSKAOYAN.COM
```

IP(返回地址) 上一层栈帧基址 若干个局部变量 栈 帧 结构 若干个调用参数 IP(返回地址) 上一层栈帧基址

```
汇编代码实战:恢复上一层栈帧
                                            0xA00F 0058
                                                           4字节
                                                                      P函数
                                                                      栈帧
caller:
                       add:
                                            0xA00F 0034
                                                           4字节
push ebp
                       push ebp
                                            0xA00F 0030
                                                        0xA00F 0058
                                                                      ebp
mov ebp, esp
                       mov ebp, esp
                                                       未初始化(sum)
                                            0xA00F 002C
                       mov eax, [ebp+12]
sub esp,24
                                            0xA00F 0028
                                                         80 (temp2)
                                            0xA00F 0024
                                                        125 (temp1)
                       mov edx, [ebp+8]
mov [ebp-12],125
                                                           4字节
                                            0xA00F 0020
                       add eax, edx
mov [ebp-8],80
                                            0xA00F 001C
                                                          80 (y)
                       leave
mov eax, [ebp-8]
                                            0xA00F 0018
                                                          125 (x)
                       ret
mov [esp+4],eax
                                            0xA00F 0014
                                                         IP (返回地址)
mov eax, [ebp-12]
                                            0xA00F 0010
                                            0xA00F 000C
mov [esp],eax
                                            0xA00F 0008
call add
mov [ebp-4],eax
                                                 eax:
                                                              edx: 125
                      int caller(){
mov eax, [ebp-4]
                                                  int add(int x, int y){
                        int templ=125;
leave
                        int temp2=80;
                                                   return x+y;
ret
                        int sum=add(templ,temp2);
                        return sum;
                                                 王道考研/CSKAOYAN.COM
```

汇编代码实战:函数调用返回 caller: add: IP(返回地址) push ebp push ebp 上一层栈帧基址 mov ebp, esp mov ebp, esp mov eax,[ebp+12] sub esp,24 若干个局部变量 mov edx, [ebp+8] mov [ebp-12],125 栈 add eax, edx mov [ebp-8],80 帧 leave mov eax, [ebp-8] 结构 ret mov [esp+4],eax mov eax, [ebp-12] mov [esp],eax 若干个调用参数 add mov [ebp-4],eax IP(返回地址) int caller(){ mov eax, [ebp-4] 上一层栈帧基址 int templ=125; leave int temp2=80; ret int sum=add(templ,temp2); return sum;

```
4字节
0xA00F 0058
                            P函数
                            栈帧
0xA00F 0034
                4字节
0xA00F 0030
             0xA00F 0058
                            ebp
            未初始化(sum)
0xA00F 002C
0xA00F 0028
              80 (temp2)
0xA00F 0024
             125 (temp1)
                4字节
0xA00F 0020
0xA00F 001C
                80 (y)
0xA00F 0018
               125 (x)
0xA00F 0014
              IP (返回地址)
0xA00F 0010
0xA00F 000C
0xA00F 0008
      eax:
                    edx:
      int add(int x, int y){
        return x+y;
```

IP(返回地址) 上一层栈帧基址 若干个局部变量 栈 帧 结构 若干个调用参数 IP(返回地址) 上一层栈帧基址

汇编代码实战: 使用返回值

```
caller:
push ebp
mov ebp, esp
sub esp,24
mov [ebp-12],125
mov [ebp-8],80
mov eax, [ebp-8]
mov [esp+4],eax
mov eax, [ebp-12]
mov [esp],eax
call add
mov [ebp-4],eax
mov eax, [ebp-4]
leave
ret
```

```
add:
push ebp
mov ebp, esp
mov eax,[ebp+12]
mov edx, [ebp+8]
add eax, edx
leave
ret
```

int caller(){

int templ=125;

int temp2=80;

return sum;

```
4字节
                      0xA00F 0058
                                                  P函数
                                                  栈帧
                      0xA00F 0034
                                      4字节
                      0xA00F 0030
                                   0xA00F 0058
                                                  ebp
                      0xA00F 002C
                                    205 (sum)
                                                  [ebp-4]
                      0xA00F 0028
                                    80 (temp2)
                      0xA00F 0024
                                   125 (temp1)
                      0xA00F 0020
                                      4字节
                      0xA00F 001C
                                      80 (y)
                      0xA00F 0018
                                     175 (x)
                                                   = esp
                      0xA00F 0014
                      0xA00F 0010
                      0xA00F 000C
                      0xA00F 0008
                            eax:
                                          edx: 125
                            int add(int x, int y){
                              return x+y;
int sum=add(templ,temp2);
                            王道考研/CSKAOYAN.COM
```

IP(返回地址) 上一层栈帧基址 若干个局部变量 栈 帧 结 若干个调用参数 IP(返回地址) 上一层栈帧基址

汇编代码实战: 传递返回值

```
caller:
push ebp
mov ebp, esp
sub esp,24
mov [ebp-12],125
mov [ebp-8],80
mov eax, [ebp-8]
mov [esp+4],eax
mov eax, [ebp-12]
mov [esp],eax
call add
mov [ebp-4],eax
mov eax, [ebp-4]
leave
ret
```

```
add:
push ebp
mov ebp, esp
mov eax,[ebp+12]
mov edx, [ebp+8]
add eax, edx
leave
ret
```

int caller(){

int templ=125;

int temp2=80;

return sum;

```
4字节
                      0xA00F 0058
                                                  P函数
                                                  栈帧
                      0xA00F 0034
                                      4字节
                      0xA00F 0030
                                   0xA00F 0058
                                                  ebp
                      0xA00F 002C
                                    205 (sum)
                                                 [ebp-4]
                      0xA00F 0028
                                    80 (temp2)
                      0xA00F 0024
                                   125 (temp1)
                                      4字节
                      0xA00F 0020
                      0xA00F 001C
                                      80 (y)
                      0xA00F 0018
                                     125 (x)
                                                   = esp
                      0xA00F 0014
                      0xA00F 0010
                      0xA00F 000C
                      0xA00F 0008
                            eax:
                                          edx: 125
                            int add(int x, int y){
                              return x+y;
int sum=add(templ,temp2);
                            王道考研/CSKAOYAN.COM
```

总结: 函数调用的机器级表示



调用者:

.....

可用 push 或 mov 指令实现

返回地址压入栈顶、 并跳转到被调用函 数第一条指令

通过eax寄存器

• 将调用参数写入当前 栈帧的顶部区域

• 执行 call 指令

• 使用返回值

....

被调用者:

保存上一层函数栈帧, 设置当前函数栈帧

• 初始化局部变量

• 一系列处理逻辑

• 向上层函数传递返回值

• 恢复上一层函数的栈帧

执行 ret 指令

push ebp mov ebp,esp 或: enter指令

[ebp-4]、[ebp-8]...

通过eax寄存器

mov esp, ebp pop ebp

或: leave指令

从栈顶找到返回地址, 出栈并恢复 IP 值

拓展: 一个栈帧内可能包含哪些内容?

IP(返回地址) 上一层栈帧基址 若干个局部变量 栈帧 结构 部分寄存器值 若干个调用参数 IP(返回地址) 上一层栈帧基址

调用其他函数前,如果有必要,可将 某些寄存器(如: eax、edx、ecx) 的值入栈保存,防止中间结果被破坏

不一定存在。如果这些寄存器值不 是运算的中间结果,则可以不保存

总结: 函数调用的机器级表示



调用者:

如 eax、edx、ecx

可用 push 或 mov 指令实现

返回地址压入栈顶、 并跳转到被调用函 数第一条指令

通过eax寄存器

保存必要的寄存器

- 将调用参数写入当前 栈帧的顶部区域
- 执行 call 指令
- 使用返回值

恢复必要的寄存器

被调用者:

保存上一层函数栈帧, 设置当前函数栈帧

• 初始化局部变量

• 一系列处理逻辑

• 向上层函数传递返回值

• 恢复上一层函数的栈帧

• 执行 ret 指令

push ebp mov ebp,esp 或: enter指令

[ebp-4]、 [ebp-8]...

通过eax寄存器

mov esp, ebp

或: leave指令

从栈顶找到返回地址 出栈并恢复 IP 值

